Financial Market Prediction Using Deep Learning Models

Gaurang Nayee
M22MA202

Under the Supervision of

Dr.Vivek Vijay & Dr.Sandeep Kumar Yadav

Department of Mathematics
IIT Jodhpur

Nov 27, 2023

Contents

Current Section

Introduction

The prediction of the financial market plays a critical role in minimizing risks faced by investors. The application of deep learning (DL) models, particularly autoencoders (AE) combined with Kernel Extreme Learning Machine (KELM), shows improved prediction accuracy. This model focuses on minimizing the prediction error for volatile financial markets such as Bank of India and other major institutions.

Current Section

Methodology

Kernel Extreme Learning Machine (KELM)

The basic ELM is a Single Layer Feedforward Neural Network (SLFN). The output of ELM with L hidden nodes is expressed as:

$$f(x) = \sum_{i=1}^{L} \beta_i h_i(x)$$

where β is the output weight vector and $h(x)$ is the feature mapping function. KELM is an extension of ELM with kernel functions for better generalization.

Kernel Extreme Learning Machine (KELM)

The output vector $\beta = [\beta_1, \beta_2, \dots, \beta_L]$ connects the hidden layer nodes (L) to the output neuron. The ELM feature mapping function is represented as:

$$h(x) = [h_1(x), h_2(x), \dots, h_L(x)]$$

Here, $x = [x_1, x_2, \dots, x_N]$ represents the input samples, where N is the number of patterns. The hidden layer parameters are randomly generated without tuning. The input data is transformed using feature mapping via an activation function:

$$h_i(x) = a(w_{i0} + w_{i1}x_{i1} + w_{i2}x_{i2} + \dots + w_{im}x_{im})$$

The hidden layer matrix is:

$$H = \begin{bmatrix} h(x_1) & \cdots & h_L(x_1) \\ \vdots & \ddots & \vdots \\ h(x_N) & \cdots & h_L(x_N) \end{bmatrix}$$

Kernel Extreme Learning Machine (KELM)

The target vector:

$$T = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

The matrix form of the problem:

$$H\beta = T$$

To optimize β and prevent overfitting:

$$L_{CELM} = \frac{1}{2}\|\beta\|^2 + \frac{1}{2C}\|\xi\|^2$$

Using the KKT theorem, β is computed as:

$$\beta = (I/C + H^T H)^{-1} H^T T \quad \text{for } N > L$$

Kernel Extreme Learning Machine (KELM)

Kernel functions can enhance ELM by mapping the data into a higher-dimensional space:

- Polynomial kernel: $K(x_i, x_j) = (1 + x_i^T x_j / \sigma^2)^g$
- Gaussian kernel: $K(x_i, x_j) = \exp(-\|x_i - x_j\|^2 / 2\sigma^2)$
- Sigmoid kernel: $K(x_i, x_j) = \tanh(b x_i^T x_j + c)$
- Wavelet kernel: $K(x_i, x_j) = \cos(d\|x_i - x_j\|^e) \exp(-\|x_i - x_j\|^2 / f)$

The predicted output:

$$f(x) = K(x, X)^T (I/C + K_N^T K_N)^{-1} K_N^T T$$

KELM Architecture

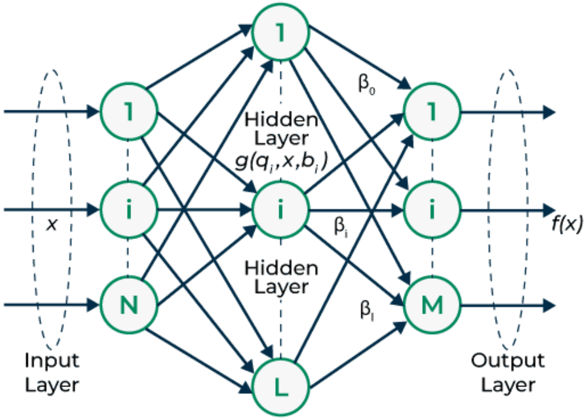


Figure 1: Architecture of Kernel Extreme Learning Machine (KELM)

Stacked Autoencoder (SAE)

SAE is formed by stacking multiple Autoencoders. Each layer learns representations, and the output from one layer becomes the input to the next. This hierarchical structure improves the model's ability to learn complex data patterns.

Stacked Autoencoder Architecture

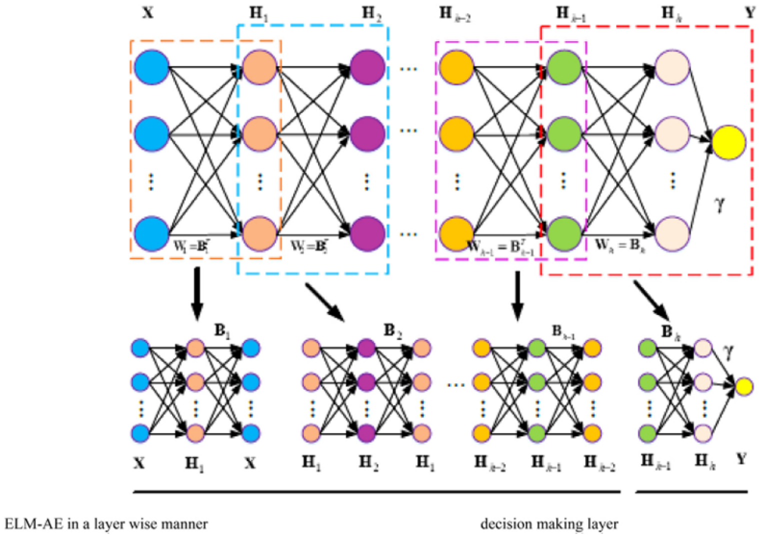


Figure 2: Architecture of SAE

Hierarchical ELM

Combines the feature extraction ability of SAE and the decision-making capability of ELM. The architecture of AE-ELM reduces errors between input and output, resulting in more accurate financial market predictions.

Multi-layer Hierarchical KELM-AE

KELM-AE enhances the prediction accuracy by incorporating kernel functions in the ELM, improving stability and reducing the need for hidden layer selection. The final layer uses a kernel matrix for decision-making.

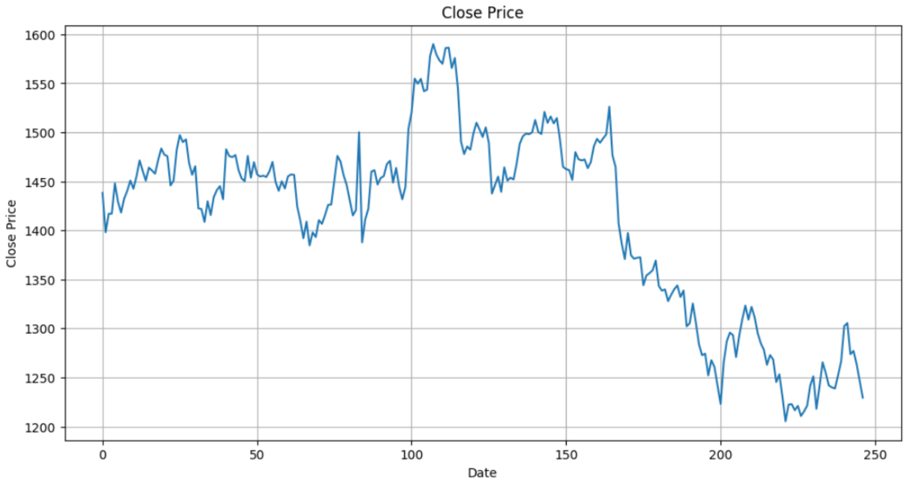
Current Section

Algorithm for Proposed Model

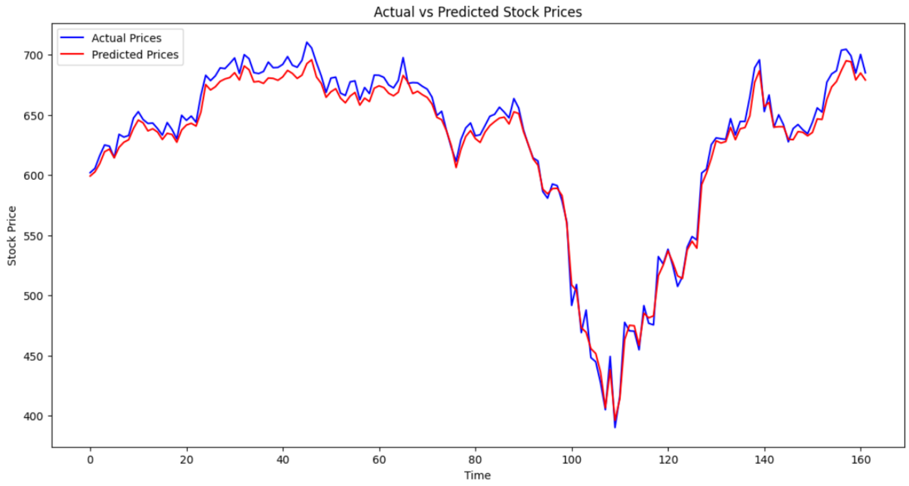
```
Input: Financial data (OHLC values)
Output: Predicted closing price
  Normalize data
  Train Stacked Autoencoder for feature extraction
  Apply Kernel Extreme Learning Machine for prediction
  Compute error using MAPE, MAE, RMSE
  return Predicted price
```

Current Section

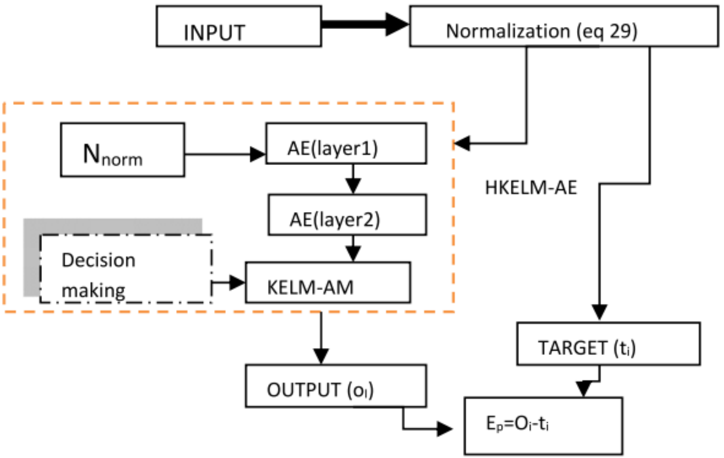
Reliance Industries Ltd, last one year



Original 24h based closing price for Reliance Industries Ltd



Prediction comparison for Reliance Industries Ltd



(a)

Model working

Flow of the Algorithm

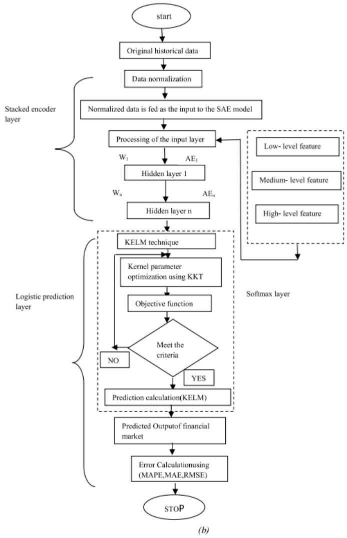
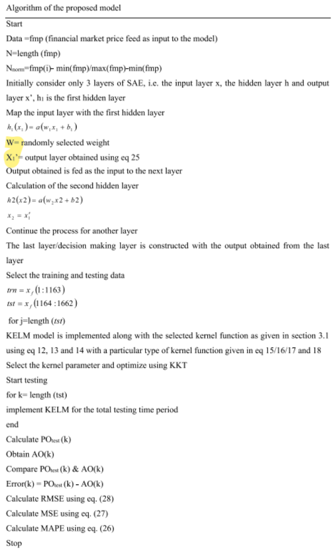

start
Original historical data
Data normalization
Stacked encoder layer
Normalized data is fed as the input to the SAE model
Processing of the input layer
W1
AE1
Hidden layer 1
Wn
AEn
Hidden layer n
Low- level feature
Medium- level feature
High- level feature
Softmax layer
KELM technique
Kernel parameter optimization using KKT
Objective function
Logistic prediction layer
Meet the criteria
NO
YES
Prediction calculation(KELM)
Predicted Outputof financial market
Error Calculationusing (MAPE,MAE,RMSE)
STOP
(b)

Figure 3: Flow of the algorithm

the Algorithm



Algorithm of the proposed model

```
Start
Data =fmp (financial market price feed as input to the model)
N=length (fmp)
$N_{norm}$=fmp(i)- min(fmp)/max(fmp)-min(fmp)
Initially consider only 3 layers of SAE, i.e. the input layer x, the hidden layer h and output layer x', $h_1$ is the first hidden layer
Map the input layer with the first hidden layer
$h_1(x_1) = a(w_1 x_1 + b_1)$
W= randomly selected weight
$X_1$'= output layer obtained using eq 25
Output obtained is fed as the input to the next layer
Calculation of the second hidden layer
$h2(x2) = a(w_2 x2 + b2)$
$x_2 = x_1'$
Continue the process for another layer
The last layer/decision making layer is constructed with the output obtained from the last layer
Select the training and testing data
$trn = x_f(1:1163)$
$tst = x_f(1164:1662)$
 for j=length ($tst$)
KELM model is implemented along with the selected kernel function as given in section 3.1 using eq 12, 13 and 14 with a particular type of kernel function given in eq 15/16/17 and 18
Select the kernel parameter and optimize using KKT
Start testing
for k= length (tst)
implement KELM for the total testing time period
end
Calculate $PO_{test}$(k)
Obtain AO(k)
Compare $PO_{test}$(k) & AO(k)
Error(k) = $PO_{test}$(k) - AO(k)
Calculate RMSE using eq. (28)
Calculate MSE using eq. (27)
Calculate MAPE using eq. (26)
Stop
```

Figure 4: pseudo algorithm

Current Section

Performance Evaluation

The performance of the proposed model is measured using:

- Mean Absolute Percentage Error (MAPE):

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{t_i - o_i}{t_i}\right| \times 100$$

- Mean Absolute Error (MAE):

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|t_i - o_i|$$

- Root Mean Square Error (RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(t_i - o_i)^2}$$

where N is the number of data points, t_i is the target, and o_i is the predicted value.

Performance Evaluation Metrics

Metric	Value
Mean Absolute Percentage Error (MAPE)	1.91%
Mean Absolute Error (MAE)	0.7053
Root Mean Square Error (RMSE)	.8390
R-squared	0.9969

Table 1: Performance Metrics for the Model on BOI Close price prediction

Metric	Value
Mean Absolute Percentage Error (MAPE)	0.18%
Mean Absolute Error (MAE)	0.4252
Root Mean Square Error (RMSE)	0.5891
R-squared	0.9999

Table 2: Performance Metrics for the Model on SBI Close price prediction

Performance Evaluation Metrics

Metric	**Value**
Mean Absolute Percentage Error (MAPE)	2.49%
Mean Absolute Error (MAE)	0.8425
Root Mean Square Error (RMSE)	.9175
R-squared	0.9958

Table 3: Performance Metrics for the Model on YES Bank Close price prediction

Metric	**Value**
Mean Absolute Percentage Error (MAPE)	0.10%
Mean Absolute Error (MAE)	1.2453
Root Mean Square Error (RMSE)	1.6052
R-squared	0.9960

Table 4: Performance Metrics for the Model on Reliance Industries Ltd Close price prediction

Current Section

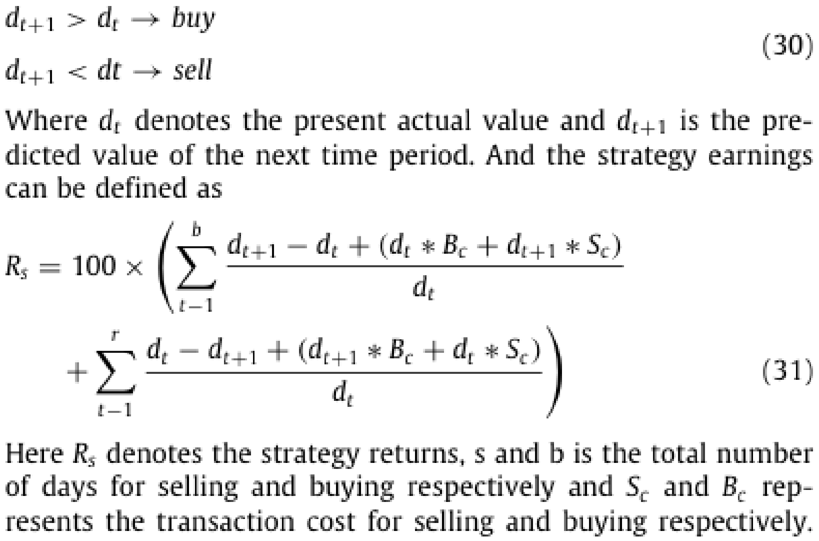
Buy/Sell Signals and Strategy Returns

The model generates Buy/Sell signals based on predicted and actual prices, helping guide trading decisions. The Buy signal is triggered when the predicted price is greater than the actual price, while the Sell signal is triggered when the predicted price is less than the actual price.

- **Buy/Sell Signals:True V.S Pred** ['Buy 1222.4471356765716 1222.300048828125', 'Buy 1225.0990310279358 1222.75', 'Buy 1217.3510070306675 1216.550048828125', 'Buy 1222.1579194724382 1221.050048828125', 'Buy 1212.3290620072867 1210.699951171875', 'Sell 1215.3314853941674 1215.449951171875', 'Sell 1221.1349469943743 1221.25', 'Sell 1240.4168392041947 1241.800048828125', 'Sell 1250.9874138812484 1251.1500244140625', 'Buy 1222.5771389846996 1218.0', 'Sell 1239.9939828833394 1240.8499755859375', 'Sell 1264.3426351688931 1265.5', 'Sell 1254.1398521585024 1254.75', 'Sell 1241.6245415583007 1241.9000244140625', 'Sell

Strategy Returns

Strategy Returns: The strategy returns for the Buy/Sell signals generated based on the model are computed as follows: Strategy Returns: 28.941478411566933 This demonstrates the effectiveness of the model in generating profitable trading signals and achieving substantial returns.



$$d_{t+1} > d_t \rightarrow buy$$
$$d_{t+1} < dt \rightarrow sell \tag{30}$$

Where d_t denotes the present actual value and d_{t+1} is the predicted value of the next time period. And the strategy earnings can be defined as

$$R_s = 100 \times \left(\sum_{t-1}^{b} \frac{d_{t+1} - d_t + (d_t * B_c + d_{t+1} * S_c)}{d_t} + \sum_{t-1}^{r} \frac{d_t - d_{t+1} + (d_{t+1} * B_c + d_t * S_c)}{d_t} \right) \tag{31}$$

Here R_s denotes the strategy returns, s and b is the total number of days for selling and buying respectively and S_c and B_c represents the transaction cost for selling and buying respectively.

Figure 5: Strategy for buying and selling

Current Section

What's next

- **Data Collection and Preprocessing**: Set up real-time data pipelines using financial market APIs like Alpha Vantage, Quandl, or broker APIs.
- **Model Adaptation for Live Trading**: Modify the deep learning model to handle real-time inputs and generate trading signals.
- **Backtesting the Strategy**: Backtest the model using historical data to evaluate its performance.
- **Broker Integration**: Integrate with a broker's API (e.g., Interactive Brokers, Alpaca) to place live orders.
- **Develop the Trading Bot**: Write code to handle order execution, risk management, and trade monitoring.
- **Paper Trading and Simulation**: Test the bot with paper trading to simulate real-market conditions.
- **Monitoring and Optimization**: Continuously monitor the botâs performance and optimize the model.
- **Live Trading Deployment**: Deploy the bot for live trading in a real market with a low-latency setup.

References

- Mohanty, D. K., Parida, Ajaya Kumar, Khuntia, Shelly Suman. *Financial Market Prediction Under Deep Learning Framework Using Autoencoder and Kernel Extreme Learning Machine.* School of Computer Engineering, KIIT Deemed to Be University, Bhubaneswar, Odisha, India.

Thank You